# An Approach for Guiding Colliding Physically-Based Models

Alexis Lamouret, Marie-Paule Cani

# An Approach for Guiding Colliding Physically-Based Models

Alexis Lamouret
Marie-Paule Gascuel

iMAGIS / IMAG, BP 53
F-38041 Grenoble cedex 09, France
email: Alexis.Lamouret@imag.fr,
Marie-Paule.Gascuel@imag.fr

January 25, 1995

**Abstract**

This paper describes a method for controlling physically-based models submitted to collisions and contacts with their synthetic environment. The aim is not to make an object follow a predefined trajectory, but to use a physical simulation to find a "correct" motion, close to the user-defined path but consistent with the object mass, inertia tensor, with the path complexity and with the casual events such as collisions occurring during motion. To achieve this, objects are provided with actuators (such as motors or muscles) of limited power, and try, at each animation step, to move towards a target position located on the given path. The successive target positions are computed from the object motion. If an object is deflected by a collision, the target position may remain the same during a few time steps, while it changes quickly along the curve when the object goes fast. Applications of this approach to the production of animations sequences where physical models are used but where objects must perform a predefined script are presented.

## 1 Introduction

Over the past few years, physically-based models [**?**, **?**, **?**, **?**] have demonstrated their usefulness for the automatic generation of realistic motions, including in collision and contact situations [**?**, **?**, **?**, **?**]. However, these models are not often used by designers. Basically, finding the set of applied forces and torques (and their variation over time) which makes an object perform a given motion is not easy. In consequence, motion control has become a very important issue in physically-based animation.

Constraint methods [**?**, **?**, **?**] and inverse dynamic techniques [**?**] enable to combine physical simulation with a predefined motion for some of the objects degrees of freedom. Very interesting experiments have been made on interactive guiding of land-vehicles [**?**]. Each vehicle is pulled by a target, which is a physical object that the user manipulates in real-time through a force-feedback device. General frameworks for integrating both direct manipulation and off-line control (such as constraints or inverse dynamics) have been presented [**?**]. Complex control modules have been developed for legged locomotion [**?**, **?**], where specific problems such as organizing the legs actions and maintaining the balance must be addressed. These modules are combined with a dynamic simulator. Optimization techniques [**?**, **?**, **?**] offer global space-time control over trajectories, but are not easy to combine with automatic collision detection and response. In [**?**, **?**], the user pre-defines interactions as extra constraints between objects holding during given time intervals. This can be time consuming (imagine, for instance, the complexity of defining constraints for a

deformable wheel rolling and jumping on a bumpy floor). Moreover, the contact characteristics – deformations of the solids, contact duration – are then computed from the user-defined constraints rather than from the physical parameters, which can affect the realism of motion.

This paper describes a general method for guiding physically-based models according to a predefined script, while preserving their ability to automatically detect and respond to collisions. Instead of enforcing the objects to follow predefined paths, we use the physical simulation to find trajectories which are close to these paths, but which are correct with respect to the objects physical characteristics (mass, inertia tensor, actuator power) and to the events such as collisions and contacts occurring during motion. In particular, the speed variations are not predefined but directly given by the physical simulation. This approach can be applied to the generation of physically-based animations sequences from geometric trajectories defined by designers. The fact that the trajectory will be adequately modified in collision and contact situations greatly simplifies the designer's task.

Section 2 discusses the basic principles of our approach. Section 3 presents a method for computing "target positions" for each object on the user-defined trajectories. Section 4 explains how the objects use these target positions to derive their actuator forces, and how they correct these forces in order to compensate from the deviation due to externally-applied actions. Section 5 describes applications of our system to the generation of an animation sequences where a script was specified, but where speed and path variations due to collisions had to be computed during the simulation. Section 6 gives conclusion and discusses work in progress.

# 2 Principles of our Approach

The aim of this work is to provide a general method to facilitate the use of physically-based modeling in the generation of animation sequences. As usual, the designer specifies key-frames to guide motion. Then, during a forwards simulation through time, the system automatically computes actuator forces and torques which make the objects perform the desired motion. During this process, the predefined trajectories are slightly corrected according to the objects physical parameters and to the maximal distance specified by the user between pre-defined and final trajectories.

When defining the initial path, the user does not have to specify precisely the speed variations, the exact trajectory, nor the deformations of the solids due to collisions or lasting contacts with other objects. All these parameters, which would be difficult to pre-define correctly, are computed during the simulation.

Suppose for instance that a goal trajectory specifies that a bird must fly to the left. Then, the system will generate different motions according to the bird's mass and inertia, to the power of its actuators, and to the casual events it may run across. In particular, the bird's motion will be slowed down and its trajectory will slightly change if someone throws objects against it (our results with this bird example will be detailed in Section 5).

## 2.1 Animation Background

Our method is developed within the framework of the animation system described in [**?**, **?**]. Objects can either be rigid or deformable. Our deformable model, based on an implicit formulation, generates exact contact surfaces between colliding objects and provides precise evaluation of response forces. Complex articulated structures can be built by specifying geometric constraints between independent objects. At each animation step, objects are first animated as if they were independent, and then their motion is corrected by the "displacement constraints" module [**?**] in order to meet constraints.

The remainder of this paper describes a method for controlling the motion of isolated objects. In our framework, the application to articulated structures is straightforwards: one just has to

define constraints connecting the objects together. If the goal trajectories independently defined for connected components are incoherent with respect to joint-constraints, the "displacement constraints" module automatically restores constraints during the simulation (see the bird simulation in Section 5).

### 2.2 Basic Choices

Figure 1: Actuated object guided guided along a path

As emphasized in the Introduction, forwards simulation over time seems to be the most convenient approach for taking collision forces into account. Dragging some parts of the objects along the user-defined paths would not be a good idea, because these paths can be incorrect with respect to dynamic laws. We prefer to associate actuators (which represent motors or muscles capable of generating forces or torques) to some of the objects. Spline curves derived from the user-defined key-frames are associated with each actuated object. Then, at each animation step, actuator forces and torques are computed from target positions located on these curves (see Figure 1).

Most of the time, a set of external actions (such as gravity, contact or response forces) are acting of the objects. Then, just applying a force towards the target position would not be sufficient. In particular, gravity would quickly bring the object far below the initial path. To cope with this problem, we provide objects with "sensors" which give them the current target position, but also the variation between their own current position and the position they expected to reach at the last time step. Thank to these sensors, each object can approximate the set of external forces acting during the last time intervall, and use this information for correcting its actuator forces.

Once the actuator forces and torques are calculated, the simulator runs to find out new positions for the objects. The description of the animation algorithm is given in Figure 2.

The next two sections detail the key points of our method: how to compute a target position on the predefined path and how to derive the actuator forces from this position and from observed external actions.

## 3 Computing a Target Position

Two kinds of actuators are available in our system. Translation actuators generate a translation force $\vec{F}$ applied to the object's center of mass, verifying the constraint: $||\vec{F}|| < F_{\max}$. Rotation actuators generate a torque $\vec{\mathcal{M}}$ verifying: $||\vec{\mathcal{M}}|| < \mathcal{M}_{\max}$.

Consequently, two different kinds of predefined trajectories, derived from the user-defined key-frames, can be associated with actuators. For translation actuators, trajectories are defined by Cardinal splines in the 3D world space. For rotation actuators, key rotation are converted into quaternions and then interpolated with Catmull-Rom splines on the 4D unit sphere. In both cases, the target position to be computed will be represented by a parameter value along the curve.

Figure 2: The animation algorithm

Moving the target position at a constant speed would not be a good solution. Indeed, the actuated object can be slowed down by a collision, and it will not follow the predefined path with a sufficient precision if the target has run too far.

We rather try to always keep the target at the same distance (called "basic distance") from the object, with the constraint that the target always moves forwards along the path. Then, if an object is pushed backwards, its target position just remains the same during a few time steps. If the solid goes fast, so will do the target.

In practice, the "basic distance" is pre-specified by the user. It also represents the maximal distance between any point of the pre-defined path and the final trajectory.

## 3.1 Target position associated with a translation actuator

Before the animation, the target is initialized at the zero-parameter on the curve. Then, at each animation step, computing a new target position starts with comparing the current distance between object and target with the basic distance. If the object is currently too far from the target position, this position is not recomputed: the target will wait until the object comes close enough. On the opposite, if the distance between object and target is smaller than the basic distance, a new target position is computed further on the spline by binary search, until a position at the basic distance from the object is found.

### 3.2 Target postion associated with a rotation actuator

First of all, let us briefly explain how we interpolate between key-orientations, in order to define a spline curve for the target.

Key positions in rotation are defined by the user as a set of axes and angles, which are converted in quaternions [**?**]. We use Catmull-Rom Splines [**?**] to interpolate between quaternions (detail about the method can be found in [**?**]).

Let $q_1$ and $q_2$ be two quaternions, and $u$ the parameter between 0 to 1, and define a *s*pherical *l*inear int*erp*olation (*slerp*) formula by:

$$q_u = slerp(q_1, q_2; u) = \frac{\sin(1-u)\theta}{\sin\theta} q_1 + \frac{\sin u\theta}{\sin\theta} q_2,$$

where $q_1.q_2 = \cos\theta$.
The Catmull-Rom spline interpolation uses *slerp* as follows:
Let $q_{00}...q_{03}$ be the control points, and $q_{30}$ the interpolated point at u, we have :

$$\begin{aligned}
q_{10} &\leftarrow slerp(q_{00}, q_{01}; u+1)\\
q_{11} &\leftarrow slerp(q_{01}, q_{02}; u)\\
q_{12} &\leftarrow slerp(q_{02}, q_{03}; u-1)\\
q_{20} &\leftarrow slerp(q_{10}, q_{11}; (u+1)/2)\\
q_{21} &\leftarrow slerp(q_{11}, q_{12}; u/2)\\
q_{30} &\leftarrow slerp(q_{20}, q_{21}; u)
\end{aligned}$$

Computing a new position for the target according to the object current orientation can be done in the same way as for translations.

First of all, we define a distance between orientations by: $d(q_a, q_b) = angle(q_a.q_b^{-1})$, where $q_a$ and $q_b$ are quaternions. The user specifies an angle giving the desired basic distance between orientations.

At each animation step, the target orientation may either remain unchanged if the object orientation is too different, either be recomputed by binary search, in order to find a new orientation further on the path which is at the basic distance from the object current orientation.

# 4 Computing Motor Forces

## 4.1 Towards the target position

The principle of the actuator action is to move towards the associated target position (or orientation). Nevertheless, trying to reach it in one time step would not be a good choice. The object would next be dragged by its inertia off the other side of the goal trajectory. Our method consists in computing an actuator force (or torque) which, in the absence of external actions, produces a displacement towards the target, reducing the distance to it by a given percentage $\alpha$.

Let $\vec{x}(t)$ be the current object position, and $\vec{v}(t)$ its speed. Under an actuator force $\vec{F}$, the object motion during a time interval $dt$ is given by:

$$\vec{v}(t+dt) = \vec{v}(t) + \frac{\vec{F}(t)}{m}\, dt \quad (1)$$

$$\vec{x}(t+dt) = \vec{x}(t) + \vec{v}(t)\, dt + \frac{1}{2}\frac{\vec{F}(t)}{m}\, dt^2 \quad (2)$$

To reach the new position $\vec{x}(t+dt) = \vec{x}(t) + \alpha\,(\vec{x}(target) - \vec{x}(t))$, the applied force must be:

$$\vec{F}(t) = 2m\,\frac{\alpha(\vec{x}(target) - \vec{x}(t))/dt - \vec{v}(t)}{dt} \tag{3}$$

Similarly, the second Euler equation of motion for a solid of inertia tensor $J$ and of current orientation matrix $R(t)$ gives:

$$\vec{\omega}(t+dt) = \vec{\omega}(t) + J^{-1}\left(\vec{\mathcal{M}}(t) - \vec{\omega}(t) \wedge J\vec{\omega}(t)\right)\,dt \tag{4}$$

$$R(t+dt) = (I + dt\,\tilde{\omega}(t+dt))\,R(t) \tag{5}$$

Then, the actuator torque which must be applied to rotate by a percentage $\alpha$ towards a rotation target orientation[1] is:

$$\vec{\mathcal{M}}(t) = \frac{J\,(\alpha\,\vec{\omega}(t+dt) - \vec{\omega}(t))}{dt} - \vec{\omega}(t) \wedge J\vec{\omega}(t) \tag{6}$$

$$\text{where}\quad \tilde{\omega}(t+dt) = \frac{R_{\text{target}} R^{-1}_{\text{object}} - I}{dt} \tag{7}$$

## 4.2 Compensating from Observed External Actions

In practice, the objects will be submitted to various external actions during an animation sequence. To stay close from the initial path, objects must take these actions into account while computing actuator forces and torques. Typically, an object submitted to a constant gravity force must compensate for it with its actuator action instead of falling down far below the goal trajectory.

A first solution would be to add to actuator actions the opposite of the set of external forces currently applied on the object. This would most of the time (ie. when the actuator has enough power) completely suppress the effect of external actions.

Rather than this solution, we prefer to provide each object with a "sensor", which doesn't give it the exact current values of external forces and torques, but give it its own current position, which can be compared with the position the object was trying to reach at the last time intervall (which we call the "predicted position"). With this method, we model objects which are deflected by sudden collisions, but which correct their actuator forces to cope with external actions which last in time, a little bit like a cosmonaut arriving to the moon who finds-out how to walk under lower gravity, but is still affected by sudden collisions.

More precisely, each object computes an approximation of the set of external forces during the last time step from the difference between the predicted position and the current position given by its sensors:

$$\vec{F}_{\text{ext}} = -2m\frac{(\vec{x}(t) - \vec{x}_{\text{predicted}}(t))}{dt^2} = \vec{F}(t-dt) - m\frac{\vec{v}(t) - \vec{v}(t-dt)}{dt} \tag{8}$$

$$\vec{\mathcal{M}}_{\text{ext}}(t) = \vec{\mathcal{M}}(t-dt) - J\left(\frac{\vec{\omega}(t) - \vec{\omega}(t-dt)}{dt}\right) + \vec{\omega}(t) \wedge J\vec{\omega}(t) \tag{9}$$

To overcome these observed external actions, the object adds a correction term to its actuator forces and torques, equal to the opposite of $\vec{F}_{\text{ext}}$ and $\vec{\mathcal{M}}_{\text{ext}}$. Then, actuator forces and torques are truncated if necessary according to the maximal available power.

[1] $\tilde{\omega}$ is the rotation speed matrix associated with the rotation speed vector $\vec{\omega}$, characterized by: $\forall \vec{a}\ \ \tilde{\omega}\vec{a} = \vec{\omega} \wedge \vec{a}$.

# 5 Samples of animations

## 5.1 Bird Flight

Figure 3 shows experiments on motion for an articulated bird. The bird is composed of a rigid body and of two rigid wings, connected to the body by hinges with angle constraints. The body is provided with translation and rotation actuators, and the wings with rotation actuators only. Thanks to the "displacement-constraints" module maintaining joints constraints between the body and the wings, no translation actuator is needed for the wings. Up and down key-rotations are given for the wings orientations in order to simulate the bird flight.

The different experiments show how the bird trajectory is modified by various collisions.

Figure 3: Bird flight variations according to different external actions

## 5.2 Simply Implicit

Figure 4, 5 and 6 are taken from the animation "Simply Implicit" presented at Siggraph'93 papers session. In this example a ball enters a scene composed of a shelf with a set of toys, collides each toy in order to make it fall, and goes away. We see that each collision of the ball deviates it from its trajectory because of reaction forces of the objects it meets, affecting also its speed.

Figure 4: "Simply Implicit": controlling a ball motion in order to throw toys to the floor. Automatic collision detection and response is used during this simulation.

# 6 Conclusion

The method developed here should ease the use of physically-based animation in situations where a script is specified. The designer controls motion by predefining basic trajectories for each object. During this process, he does not have to consider at all the dynamic properties of the objects nor the possible collisions and contacts that will take place during motion: Our system will automatically *correct* the paths according to all these parameters.

Deflected by unexpected external actions, the objects try to adapt their actuator forces in order to come back closer to their goal trajectory. Instead of being dictated by the user, the speed variations during motion depend on the power of the object's actuators, on its mass and inertia, on the trajectory complexity, and on the casual events such as collisions which can accelerate the motion or slow it down.

In the current implementation, target trajectories are independently defined for each solid component, so there is no synchronization between the different actuators acting on an articulated body, nor between different body's motions. Work in progress includes attempts to use a finite-state graph layer to add synchronization constraints to the system.

# References


[AG85] W. Armstrong and M. Green. The dynamics of articulated rigid bodies for purposes of animation. In *Graphics Interface 85*, pages 407–415, Montréal, May 1985.

[Bar92] David Baraff. Dynamic simulation of non-penetrating rigid bodies. *PHD Thesis*, Cornell University, May 1992.

[BB88] R. Barzel and A. Barr. A modeling system based on dynamic constraints. *Computer Graphics*, 22(4):179–188, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, August 1988).

[BG88] P.J. Barry and R.N. Goldman. A recursive evaluation algorithm for a class of catmull-rom splines. *Computer Graphics*, 22(4):199–204, August 1988.

[BN88] L. Shapiro Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, August 1988).

[Coh92] M. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992. Proceedings of SIGGRAPH'92 (Chicago, Illinois, July 1992).

[Fea83] R. Featherstone. The calculation of robot using articulated-body inertias. *International Journal of Robotics Research*, 2(1):13–30, 1983.

[Gas93] Marie-Paule Gascuel. An implicit formulation for precise contact modeling between flexible solids. *Computer Graphics*, pages 313–320, August 1993. Proceedings of SIGGRAPH'93 (Anaheim, California, August 1993).

[GG92] J.D. Gascuel and M.P. Gascuel. Displacement constraints: A new method for interactive dynamic animation of articulated solids. In *Third Eurographics Workshop on Animation and Simulation*, September 1992. To appear in The Visual Computer, 1993.

[Hah88] J.K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[IC87] P.M. Isaacs and M.F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics*, 21(4):215–224, July 1987. Proceedings of SIGGRAPH'87 (Anaheim, California, July 1987).

[JLR93] S. Jimenez, A. Luciani, and O. Raoult. Physical simulation of land vehicles with obstacle avoidance and various terrain interactions. *The Journal of Visualisation and Computer Animation*, 4:79–94, 1993.

[LAH93] Ch. Lecerf, B. Arnaldi, and G. Hegron. Mechanical systems motion control for computer animation. *Proceedings ATARV'93*, July 1993.

[LJF+91] A. Luciani, S. Jimenez, J.L. Florens, C. Cadoz, and O. Raoult. Computational physics: a modeler simulator for animated physical objects. *Proceedings of the Eurographics Conference*, September 1991.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. *Computer Graphics*, 22(4):289–298, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, August 1988).

[Ove91] C. Van Overveld. An iterative approach to dynamic simulation of 3-D rigid-body motions for real-time interactive computer animation. *The Visual Computer*, 7:29–38, 1991.

[RH91] M. Raibert and J. Hodgins. Animation of dynamic legged locomotion. *Computer Graphics*, 25(4):349–358, July 1991. Proceedings of SIGGRAPH'91 (Las Vegas, Nevada, July 1991).

[Sch91] J. Schlag. Using geometric constructions to interpolate orientation with quaternions. In *Graphic Gems II*, pages 377–380. James Arvo Ed, Academic Press, 1991.

[Sho85] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, July 1985.

[vdPFV92] M. van de Panne, E. Fiume, and Z.G. Vranesic. Control techniques for physically-based animation. In *Third Eurographics Worshop on Animation and Simulation*, Cambridge, England, September 1992.

[WB85] J. Wilhems and B. Barsky. Using dynamic analysis to animate articulated bodies as humans and robots. In *Graphics Interface 85*, pages 97–104, Montréal, May 1985.

[WK88] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988. Proceedings of SIGGRAPH'88 (Atlanta, August 1988).